Python - Spatial Analysis (Intermediate)

Jason Pardy & Nobbir Ahmed
Today’s Agenda

- Builds on *Python – Getting Started*
- **Walkthrough** – Create a analytical tool to do Quadrat Analysis
  1. Python 201: data structures & functions
  2. Script tools
  3. Creating and using classes for parameter values
  4. Accessing data with cursors
  5. Making scripts more efficient

- More on cursors and working with geometries
ArcPy

- The access point to geoprocessing tools

- A package of functions, classes and modules, all related to scripting in ArcGIS
  - Functions that enhance geoprocessing workflows (ListFeatureClasses, Describe, SearchCursor, etc)
  - Classes that can be used to create complex objects (SpatialReference, FieldMap objects)
  - Modules that provide additional functionality (Mapping, SpatialAnalyst modules)

- Extends on arcgisscripting module (pre-10.0)
## Take advantage of key Python data structures

<table>
<thead>
<tr>
<th>Type</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>Flexible ordered collection</td>
<td><code>L = ['10 feet', '20 feet', '50 feet']</code></td>
</tr>
<tr>
<td>Tuple</td>
<td>An immutable list (not editable)</td>
<td><code>T = ('Thurston', 'Pierce', 'King')</code></td>
</tr>
<tr>
<td>Dictionary</td>
<td>Key/value pairs</td>
<td><code>D = {&quot;ProductName&quot;: &quot;desktop&quot;, &quot;InstallDir&quot;: &quot;c:\ArcGIS\Desktop10.0&quot;}</code></td>
</tr>
</tbody>
</table>
List comprehensions

- Provides a clean and compact way of mapping a list into another list by applying a function to each of the elements of the list.

```
List Comprehension

>>> states = [' california', ' alaska', ' maine ']
>>> states2 = [state.strip() for state in states]
>>> print states2
['california', 'alaska', 'maine']
```
Defining Functions

- A simple way to organize and re-use functionality

```
import arcpy

def increase_extent(extent, factor):
    """Increases the extent by the given factor"""

    XMin = extent.XMin - (factor * extent.XMin)
    YMin = extent.YMin - (factor * extent.YMin)
    XMax = extent.XMax + (factor * extent.XMax)
    YMax = extent.YMax + (factor * extent.YMax)

    return arcpy.Extent(XMin, YMin, XMax, YMax)

oldExtent = arcpy.Describe("boundary").extent
newExtent = increase_extent(oldExtent, .1)
```
Python 2.6

- ArcGIS 10.0 supports and installs Python 2.6
- The development of Python 3.0 has influenced many features in 2.6.
  - The % operator is supplemented by a more powerful string formatting method, .format()
  - The print statement becomes the print function in 3.0.
    - Python 2.6 has a __future__ import, letting you use the functional form instead. For example:

      ```
      >>> from __future__ import print_function
      >>> print('Hello world')
      >>> print('Hello world')
      ```

- [http://docs.python.org/whatsnew/2.6.html](http://docs.python.org/whatsnew/2.6.html)
Demo – Python 201

Python Data Structures

```python
def get_value(key):
    ii = arcpy.GetInstallInfo()
    return ii.get(key)

global_dir = get_value('InstallDir')
print(f'U:\\program files (x86)\\arcgis\\desktop10.1\\')
```
Quadrat Analysis Walkthrough
Demo – the Quadrat Analysis tool – Step 1
Python & the Geor-processing Framework

- Tightly integrated
  - No UI programming
  - Script tools look & behave like core GP tools
  - Script tools provide default validation
  - Documentation of script tools is the same as core tools
Creating a script tool…

- Script tool must include input and output arguments
  - Use `GetParameterAsText()` or `GetParameter()` to obtain script argument values

- Script must include messaging
  - Return informative messages during execution of the script
  - Return error messages when a problem arises
  - Three functions to support tool messaging
    - `AddMessage()`
    - `AddWarning()`
    - `AddError()`
Script tools – Output parameters

- All tools should have an output
  - If the script updates an input dataset, create a derived parameter
  - If an output parameter is a scalar value, make it derived
    - Use `SetParameterAsText()` or `SetParameter()` functions to set it at the end of your script
    - Allows chaining of the output value in a model
Jump Start Your Python Project with a Script Template

#--------------------------------
# Name:        template.py
# Purpose:     
# Author:      
# Created:     23/06/2011
# Copyright:   (c) company name
# ArcGIS Version:  10
# Python Version:  2.6
#--------------------------------
import os
import sys
import arcpy

def main(*argv):
    """TODO: Add documentation about this function here""
    try
        #TODO: Add analysis here
        pass
    except arcpy.ExecuteError:
        print arcpy.GetMessages(2)
    except Exception as ee:
        print ee.args[0]
    # End main function

# Ensures this file is usable as a script, script tool, or as an importable module
if __name__ == '__main__':
    argv = tuple(arcpy.GetParameterAsText(i) for i in range(arcpy.GetArgumentCount()))
    main(*argv)
Quadrat Analysis script - Step 2

Jump start our analysis using a template
ArcPy Classes
Classes

- Classes can be used to create objects
- Some parameters cannot be easily defined with a string
  - Such as a spatial reference or extent
- Classes can be used to define parameters

```python
installDir = arcpy.GetInstallInfo()['InstallDir']

prj_file = os.path.join(installDir, "North America Equidistant Conic.prj")

# Create a spatial reference using a projection file
spatial_ref = arcpy.SpatialReference(prj_file)

# Run CreateFeatureclass using the spatial reference
arcpy.CreateFeatureclass_management(input_workspace, output_name, "POLYLINE", "", "", "", spatial_ref)
```
Classes cont...

• A better user experience

```python
point = arcpy.Point(25282, 43770)
point_geom = arcpy.PointGeometry(point)
arcpy.Buffer_analysis(point_geom, "in_memory/point_buffer", "250 Meters")

extent = arcpy.Describe(feature_class).extent
arcpy.CreateRandomPoints_management("c:/data", "samplepoints", ",", extent, 500)
```
Accessing Data with Cursors
## Cursors

<table>
<thead>
<tr>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SearchCursor (dataset, {where_clause}, {spatial_reference}, {fields}, {sort_fields})</td>
<td>Read-only access to field values, geometry</td>
</tr>
<tr>
<td>UpdateCursor (dataset, {where_clause}, {spatial_reference}, {fields}, {sort_fields})</td>
<td>Update or delete field values, replace geometries</td>
</tr>
<tr>
<td>InsertCursor (dataset, {spatial_reference})</td>
<td>Add new records to a table or feature class; write field values and geometry</td>
</tr>
</tbody>
</table>
Cursors

- ArcPy cursors support iteration

```python
for row in arcpy.SearchCursor(table):
    print (row.getValue("FieldName"))
```
Cursors

- Cursors have optional arguments
  - Need coordinate information in a different coordinate system?
  - Features may be projected on-the-fly using the Spatial Reference parameter

```python
# Create a SR object from a projection file
SR = arcpy.SpatialReference("c:/NAD 1983 UTM Zone 10N.prj")

# Create search cursor, using spatial reference
rows = arcpy.SearchCursor("D:/data.mdb/roads", ",", SR)
```
Quadrat Analysis – Step 2

- Extent class
- Search Cursor
- GP Tools
- Chi-square function
Performance Tips:

• Use *in_memory* workspace for intermediate datasets
  
in_memory\outputFC

• Provide a set of fields to cursor functions
  
srows = arcpy.SearchCursor(input_fc, "", "", "fieldA;fieldB")
Quadrat Analysis – Recap

• Development time was fast:
  - A script template jump started our development
  - ArcPy provided quick and easy access to tools and functions
  - Script tool meant no UI development
  - Script tools provided default validation

• Took advantage of a function to define and group a useful piece of functionality

• Took advantage of open source libraries (i.e. numpy)

• Easy to share and deploy
  - No ArcObjects or dlls to register
Reading Geometry

- Feature classes have a geometry field
  - Typically *(but not always)* named *Shape*
- Returns a geometry object
  - Has properties that describe the feature
    - area, length, isMultipart, partCount, pointCount, type, ...
- Geometry objects can often be used in place of feature classes

```python
# Buffer each feature to a new feature class
for row in arcpy.SearchCursor("C:/data/Roads.shp"):  
geom = row.getValue("Shape")
name = row.getValue("Name")

print (geom.length)
arcpy.Buffer_analysis(geom, "buffer_{0}".format(name), "100 Feet")
```
Reading Feature Geometry

• You must understand the hierarchy for geometry in order to use it
  - A feature class is made of features
  - A feature is made of parts
  - A part is made of points

• In Python terms
  - A single part feature looks like this
    \[[\text{pnt}, \text{pnt}, \text{pnt}]\]
  - A multipart polygon feature looks like this
    \[[[\text{pnt}, \text{pnt}, \text{pnt}], [\text{pnt}, \text{pnt}, \text{pnt}]]\]
  - A single part polygon feature with a hole (inner ring) looks like
    \[[[\text{pnt}, \text{pnt}, \text{pnt}, \text{None}, \text{pnt}, \text{pnt}, \text{pnt}]]\]
for row in arcpy.SearchCursor(polygonFC):
    for part in row.shape:
        pnt = part.next()
        while pnt:
            print pnt.X, pnt.Y
            pnt = part.next()
        if not pnt:
            pnt = part.next()
        if pnt:
            interiorRing = True

Loop through each row
Loop through each part in a feature
Loop through each point in a part
For polygons, watch for interior rings
Writing Feature Geometry

- Insert cursors must be used to create new features
  
  `icur = arcpy.InsertCursor("D:/data.gdb/roads")
  row = icur.newRow()

- Use Point and Array classes to create feature parts
- A part may be used to set a geometry field
  - A multipart feature is an array containing other arrays, where each array is a part

- An Update cursor can be used to replace a row’s existing geometry
Writing Feature Geometry

# Open an insert cursor for the feature class
icur = arcpy.InsertCursor(fc)

# Create array and point objects
pt_list = [arcpy.Point(358331, 5273193),
          arcpy.Point(358337, 5272830)]

line_array = arcpy.Array(pt_list)

# Create a new row for the feature class
feat = icur.newRow()

# Set the geometry of the new feature to the array of points
feat.Shape = line_array

# Insert the feature
icur.insertRow(feat)

# Delete cursor
del icur
Geometry operators

- Python geometry objects support relational operators at 10
  - contains
  - crosses
  - disjoint
  - equals
  - overlaps
  - touches
  - within
import arcpy
line1 = arcpy.Polyline(arcpy.Array([arcpy.Point(1,10), arcpy.Point(10,10)]))
line2 = arcpy.Polyline(arcpy.Array([arcpy.Point(5,5), arcpy.Point(7,15)]))

# Does line1 cross line2? crosses return a boolean
line1.crosses(line2)
Demo - Geometry operators

Finding overlaps
Learning Python with ArcGIS

- Resource Center
  - http://resources.arcgis.com/geoprocessing/

- Desktop Help

- Python Tips, Tricks, and Hacks

- Have a good Python Reference
  - “Learning Python” by Mark Lutz
    - published by O’Reilly & Associates
  - “Core Python” by Wesley J. Chun
    - published by Prentice-Hall
  - Python Standard Library by Example –
    - Doug Hellman
Review of IDEs for Python

- Which Python IDE is best?
Esri Training for Python

http://www.esri.com/training

- Instructor-Led Course
  - Introduction to Geoprocessing Scripts Using Python

- Web Course
  - Using Python in ArcGIS Desktop 10
Additional Python Technical Sessions

- **Python – Raster Analysis**
  - Wed 3:15pm Room 5 A/B

- **Using R in ArcGIS**
  - Wed 12:00pm Room 1 A/B

- **Python - Scripting for Map Automation**
  - Wed 3:15pm Room 9

- **Building Tool with Python**
  - Thu 10:15am Room 9
Questions?
Survey: www.esri.com/sessionvals